# AngularJS Dependency Injection

**Topics :** AngularJS
**Written on** January 09, 2024

In AngularJS, dependency injection is a design pattern that facilitates the management of components and their dependencies. AngularJS uses dependency injection to make components more modular, reusable, and testable. Here's an overview of dependency injection in AngularJS:

## Basics of Dependency Injection:

In AngularJS, components like controllers, services, and directives can have dependencies on other components. Dependency injection is the process of providing these dependencies to a component rather than the component creating or managing its dependencies.

AngularJS's injector system is responsible for resolving and providing these dependencies to components.

## Example of Dependency Injection in a Controller:

```
// Define a custom service named 'myDataService'
angular.module('myApp').service('myDataService', function() {
this.getData = function() {
return ['Item 1', 'Item 2', 'Item 3'];
};
});

// Inject 'myDataService' into the 'MyController' controller
angular.module('myApp').controller('MyController', function($scope, myDataService) {
$scope.items = myDataService.getData();
});
```

In this example:

- The `myDataService` service is injected into the `MyController` controller.
- AngularJS automatically resolves the dependencies and provides an instance of `myDataService` to the controller.

## Dependency Injection in Directives:

```
// Define a custom directive named 'myDirective'
angular.module('myApp').directive('myDirective', function() {
return {
restrict: 'E',
template: '<p>{{ message }}</p>',
controller: function($scope, myDataService) {
$scope.message = 'Data from directive: ' + myDataService.getData()[0];
}
};
});
```

In this example, the `myDataService` service is injected into the directive's controller.

## Dependency Injection in Services:

```
// Define a custom service named 'myOtherService'
angular.module('myApp').service('myOtherService', function() {
this.getAnotherData = function() {
return ['Item A', 'Item B', 'Item C'];
};
});

// Inject 'myDataService' and 'myOtherService' into the 'MyCombinedService' service
angular.module('myApp').service('MyCombinedService', function(myDataService, myOtherService) {
this.getAllData = function() {
return myDataService.getData().concat(myOtherService.getAnotherData());
};
});
```

In this example, the `myDataService` and `myOtherService` services are injected into the `MyCombinedService` service.

## Benefits of Dependency Injection:

1. **Modularity:**
   - Components can be developed and tested independently, making the application more modular.
2. **Reusability:**
   - Components with dependencies can be reused across different parts of the application.
3. **Testability:**
   - Easier unit testing as dependencies can be easily mocked or replaced during testing.
4. **Maintainability:**
   - Components are less tightly coupled, making it easier to modify, extend, or replace them.

## Implicit vs. Explicit Dependency Injection:

AngularJS supports both implicit and explicit dependency injection.

- **Implicit:**

    - Components declare their dependencies in the function parameters, and AngularJS resolves them automatically based on the parameter names.

```
angular.module('myApp').controller('MyController', function($scope, myDataService) {
// ...
});
```

- **Explicit:**
    - Dependencies are explicitly specified using an array syntax. This can be useful for minification where parameter names might get changed.

```
angular.module('myApp').controller('MyController', ['$scope', 'myDataService',
function($scope, myDataService) {
// ...
}]);
```

## Minification and Dependency Injection:

In minified code, AngularJS relies on the parameter names of the function to determine dependencies. If you're using implicit dependency injection, it's important to use tools like ng-annotate during the build process to handle minification.

```
angular.module('myApp').controller('MyController', function($scope, myDataService) {
// ...
});
```

After minification, this might become:

```
angular.module('myApp').controller('MyController', function(a, b) {
// ...
});
```

However, with ng-annotate, it automatically adds the necessary annotations for correct dependency injection.